Adapting CLIM Applications for Use on the World Wide Web*

Suzanne M. Paley and Peter D. Karp
Artificial Intelligence Center
SRI International
333 Ravenswood Ave., EJ231
Menlo Park, CA 94025
{paley, pkarp}@ai.sri.com

September 13, 1995

1 Abstract

The World Wide Web (WWW) offers the potential to deliver specialized information to an audience of unprecedented size. Along with this exciting new opportunity, however, comes a challenge for software developers: instead of rewriting our software applications to operate over the WWW, how can we maximize software reuse by retrofitting existing applications? We have developed a Web server tool, written in Common Lisp, that allows any existing graphical user interface application written using the Common Lisp Interface Manager (CLIM) to hook easily into the WWW. This tool — CWEST (CLIM-WEb Server Tool, pronounced "quest") — has been developed to operate with EcoCyc, an electronic encyclopedia of genes and metabolism of the bacterium *E. coli*. EcoCyc consists of a database of objects relevant to *E. coli* biochemistry and a sophisticated interface, implemented in CLIM, that runs on the local host window system and generates graphical displays appropriate to each type of object. Each query to our server is passed as a command to the EcoCyc program, which responds by dynamically generating an appropriate local drawing. That drawing, which can be a mixture of text and graphics, is then translated into the HyperText Markup Language (HTML) and/or the Graphics Interchange Format (GIF) and returned to the client. Sensitive regions embedded in the CLIM drawing are converted to hyperlinks with Universal Resource Locators (URLs) that generate further EcoCyc queries. This tight coupling of CLIM output with Web output makes CLIM an ideal high-level programming tool for Web applications. The flexibility of Common Lisp and CLIM made implementation of the server tool surprisingly easy, requiring few changes to the existing EcoCyc program. The results can be seen at URL `http://www.ai.sri.com/ecocyc/browser.html`. We plan to make CWEST available to the CLIM community at large, with the hope that it will spur other software developers to make their CLIM applications available over the WWW.

2 Introduction

The advent of the World Wide Web (WWW) presents both opportunities and challenges. Information providers now have the means of making their information available to a far larger audience than they had previously thought possible. The challenges are two: can we retrofit existing applications to operate over the WWW without a need for a major rewrite? And can we develop powerful environments for authoring new WWW applications?

We faced this challenge recently in the context of the EcoCyc project[2], a collaboration between SRI and the Marine Biological Laboratory to develop an electronic encyclopedia of *E. coli* metabolism. The EcoCyc program is a graphical interface application that allows users to query and visualize information about genes and biochemical pathways. Although EcoCyc has received positive reviews from those who have used it, distribution impediments have limited its potential audience. Potential users must invest a certain amount of time, computer resources, and technical knowledge to download and install EcoCyc on their own computers. In addition, EcoCyc is written using the Common Lisp Interface Manager (CLIM)[1], which currently runs only on Unix workstations under X-Windows — despite the fact that most biologists use PC or Macintosh systems[1]. Given these difficulties, the prospect of making EcoCyc easily available over the WWW to any biologist or biology student with an internet connection is appealing.

Rather than rewriting EcoCyc as a Web application (an extremely time-consuming task) we have developed CWEST (CLIM-WEb Server Tool, pronounced "quest"), which forms a wrapper around the existing EcoCyc application without interfering with its normal operation. The resulting program is a dedicated EcoCyc Web server, implementing that portion of the HyperText Transfer Protocol (HTTP) necessary to process EcoCyc requests. Requests received over the Web are converted to instructions that cause EcoCyc to generate appropriate graphical output. That output is captured and converted to a form that can be passed back to the Web client. This approach was quick to implement, required few changes to the EcoCyc source code, and proved highly effective. The results can be seen at `http://www.ai.sri.com/ecocyc/browser.html`. Because the EcoCyc program is still under development, our approach of merely providing a wrapper for the existing code offers the added advantage that any future enhancements to the native interface will be automatically available in the Web version. In this sense, CLIM has become a high level tool for developing EcoCyc as a WWW application.

Our approach can be generalized to make any CLIM application available over the WWW. We have broken our tool into two modules: one to use as a basis for porting any CLIM application to the WWW, and the other to provide all EcoCyc-specific commands and specializations required for the Web implementation but not part of the original EcoCyc program. Developers who wish to use CWEST with other applications will have to supply this second module, specialized for their applications. We plan to make CWEST publicly available to the CLIM community, with the hope that it will prove useful to other CLIM developers anxious to take advantage of the new opportunities afforded by the WWW.

3 CLIM Concepts

CLIM is a sophisticated high-level tool for building graphical user interfaces (GUIs). Two CLIM concepts are important to the discussion of the implementation of our Web server tool. The first is the idea of *presentations* and *presentation translators*. A presentation is logically a graphical object — an association between a piece of screen text and/or graphics, and an application object. For example, the schematic drawing of a chemical compound may be associated with EcoCyc's internal representation for that compound. The CLIM programmer may define a presentation translator that tells the program what command to execute when the user clicks on presentations of a certain type. For example, in EcoCyc, a translator has been defined so that when any presentation of type *compound* is clicked on from within the display of a chemical reaction, all information regarding that compound appears in a new display. Using presentations and presentation translators to define active regions within a display, an application can dynamically create a sophisticated assortment of hyperlinks to new displays.

The second important concept is that of output recording. CLIM maintains an internal data structure that keeps a record of all display output, organized into a hierarchy of elements called *output records*. The leaf records are the individual output elements, such as a line of text or a graphics operation. Because presentations are a specialized type of output record, all information about presentations resides within the output record hierarchy. CLIM uses output records for several tasks. First, they allow selective regions of the display to be

[1] A version of CLIM for the Macintosh exists, but it is old, buggy, and unsupported and is therefore not suitable for general distribution. We expect versions of CLIM for the PC to be available later in 1995.

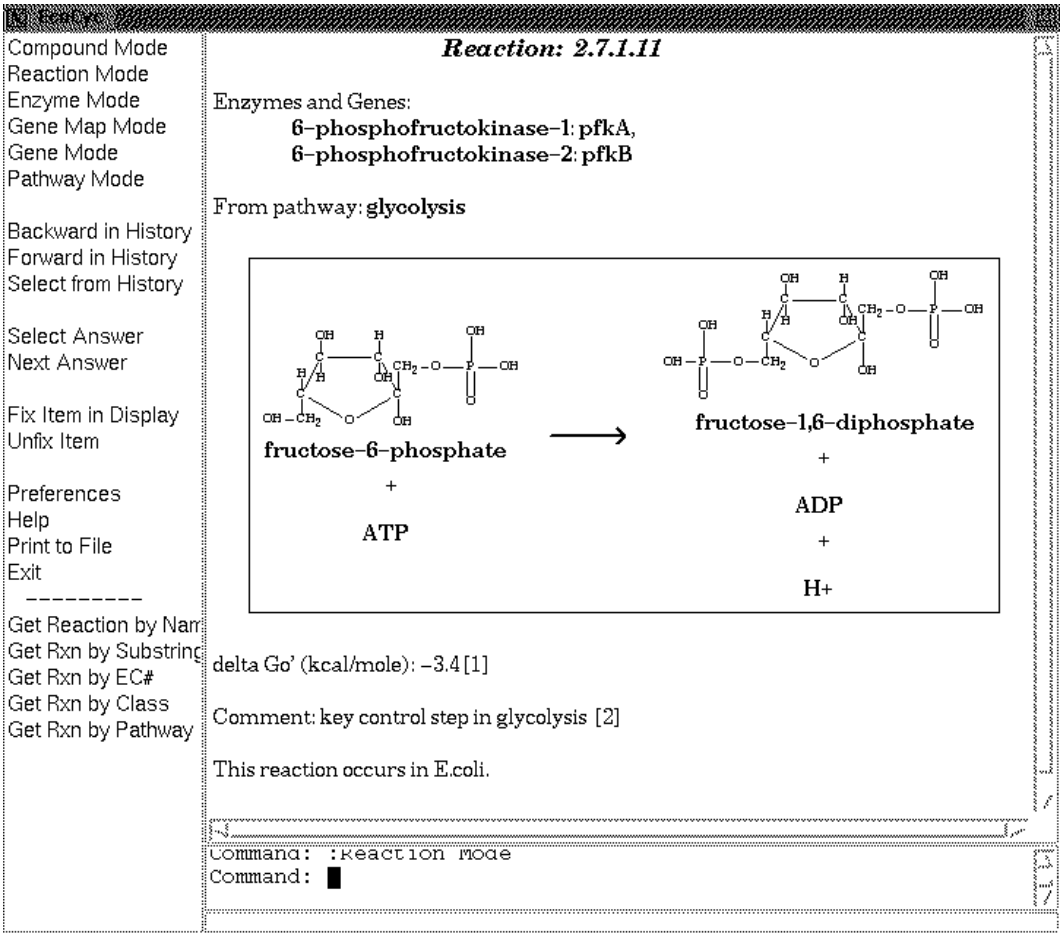

Figure 1: Sample EcoCyc display

refreshed. Second, they are used to process input events such as mouse clicks. Third, the the output record hierarchy provides a handle by which a CLIM programmer can access particular pieces of display output (to erase them, for example). This handle has been the key to the implementation of CWEST, allowing us to isolate the response of the EcoCyc program to a particular command and to break it into its various components.

4 Description of EcoCyc

EcoCyc is an ongoing project to create an encyclopedia of metabolic information for the bacterium *E. coli*. Information on biochemical compounds, reactions, enzymes, genes, metabolic pathways, and other objects has been collected and encoded in a large knowledge base. We have developed a GUI, implemented in Common Lisp and CLIM, to generate parameterized displays of information about each kind of object in the knowledge base. These displays contain part textual information — facts and descriptive comments about an object — and part graphical information, such as chemical structure drawings or metabolic pathway diagrams. Object displays, including the corresponding graphics, are computed directly from the knowledge base. Users can customize various aspects of these displays. For example, they can specify whether metabolic pathway diagrams should include compound structures or just the compound names, or whether more complex pathways should be displayed in full or in outline form.

Relationships among objects are encoded by hyperlinks using CLIM's presentation facility. Figure 1 shows a sample EcoCyc display for the reaction of the compound fructose-6-phosphate to form the compound fructose-1,6-

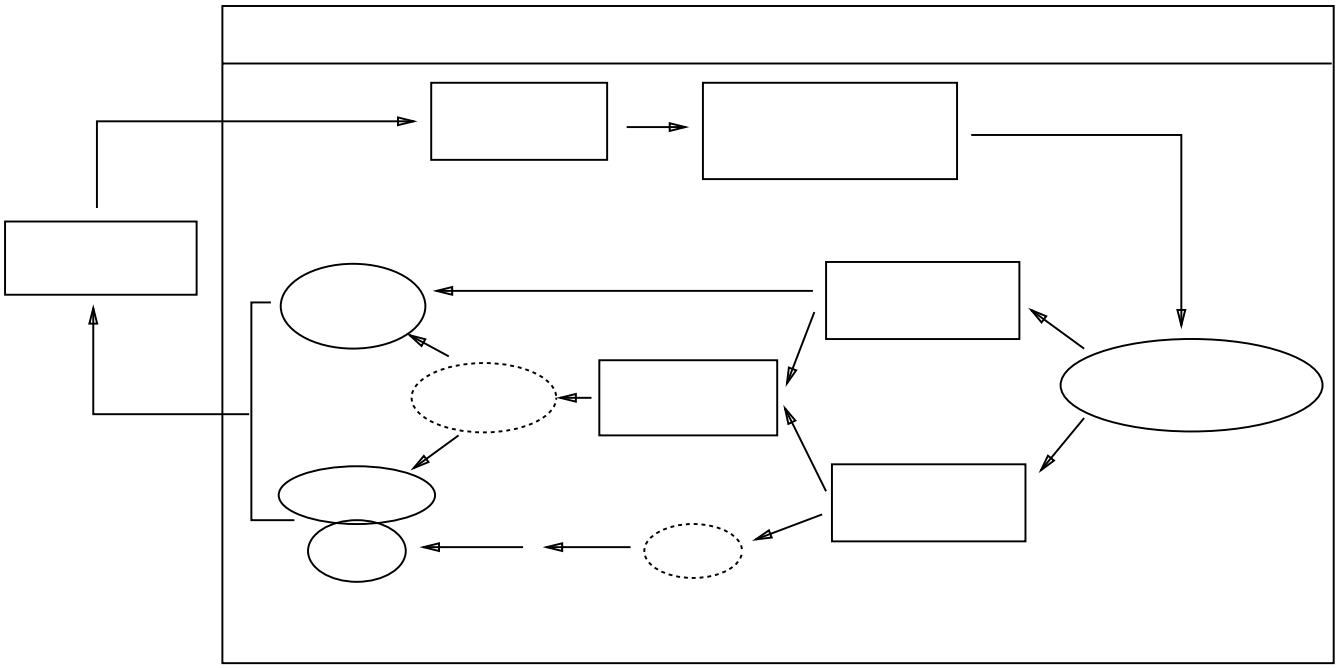

Figure 2: Dataflow diagram illustrating CWEST operation

diphosphate. Clicking on the compound name or structure for fructose-6-phosphate or fructose-1,6-diphosphate translates via a CLIM presentation translator to a command that calls up the display for that compound. Similarly, the names of enzymes, genes, and pathways, as well as the bracketed citation numbers, are all sensitive to mouse clicks. Clicking on any one of these objects navigates to the display for that object. This model is remarkably similar to the Web model, where the user can navigate to a new page by clicking on a link within the multiple linked pages. Like interconnected Web pages, object displays in EcoCyc are independent of the path taken to reach them. EcoCyc thus seems ideally suited for access over the Web. Given our desire to see the information in EcoCyc disseminated to as many biologists as possible, we had a strong motivation to translate our CLIM displays into a Web-accessible form.

5 Implementation

CWEST is a dedicated light-weight server process running in the same Lisp image as the EcoCyc application, in parallel with it. Currently, it obeys only that portion of HTTP that is necessary to process EcoCyc requests (the main SRI Web server is located on another machine).

Figure 2 outlines the processing of a Web request by the EcoCyc server. Once received, the request is converted to an instruction (a) to EcoCyc to generate a new drawing (b). The corresponding output record is captured (c) and parsed into text regions (d) and graphics regions (e). Text output records are converted directly to HTML (f). A graphics output record is converted through several steps to a GIF image (g). Presentations (h) may be embedded within both graphics and text output records, and are parsed to produce hyperlink Universal Resource Locators (URLs) (i). Text hyperlinks can be inserted directly into the HTML output. For graphics hyperlinks, an *image map* (j) must be generated that maps each hyperlink to its position in the GIF image. Finally, the HTML output with the mapped inline GIF image is returned to the Web client.

An incoming query, expressed as a URL, is parsed to generate a function call to the EcoCyc program. The incoming URL consists of four parts: the server address, the command name, the command arguments, and additional preferences. The server address is ignored once the query reaches the server. A

Display type	Before			After		
	Image dimensions	GIF file size	Time	Image dimensions	GIF file size	Time
Compound	897x615	14K	3.8s	422x166	2K	1.0s
Reaction	541x547	7K	2.3s	497x227	2K	1.3s
Enzyme	625x1042	18K	4.9s	none	0	0.8s
Small Pathway	503x674	5K	2.9s	462x530	3K	2.4s
Large Pathway	551x1816	24K	9.5s	550x1438	14K	8.0s

Table 1: Sample timings for typical object displays in various categories

dispatch function ensures the command given in the URL is a valid command, and maps it to the appropriate top-level EcoCyc function. Each top-level function takes exactly two arguments, a type and an object. These arguments are usually required and are supplied as variables in the URL. Any additional variables in the URL are optional and are parsed as preferences, which are used to modify the EcoCyc environment while the command is being executed. The preference part of the URL is appended to each additional EcoCyc URL generated in the course of creating the display, so that the same preferences will remain in effect for future displays. An example URL is the one used to display the reaction shown in Figures 1 and 4: `http://ecocyc.ai.sri.com:1555/new-image?type=REACTION&object=6PFRUCTPHOS-RXN`. `http://ecocyc.ai.sri.com:1555/` is the address of the EcoCyc server. `new-image` is the command name, instructing EcoCyc to create a new object display. The object to be displayed is `6PFRUCTPHOS-RXN`, which is the internal key for this particular reaction, and its presentation type is `REACTION`. No preferences are indicated in this example.

After receiving and parsing a query, the designated EcoCyc function is invoked to produce the desired display. In our first version of the tool, the entire output record for the resultant CLIM display was captured and converted to a GIF image. This process requires no changes to the existing EcoCyc program. The output record is then searched for presentations. For each presentation type, we call a procedure that generates an appropriate URL for the hyperlinked command. For example, the presentation of a compound within a reaction display will result in a URL that queries the EcoCyc server for the corresponding compound display. An *image map* created for the GIF image lists the bounding box of each sensitive region, which is calculated from the bounding box of the presentation and the position of the parent output record in the EcoCyc display, and its associated URL. The final document returned to the Web client is an HTML document containing the inlined GIF image, along with a link to the image map.

A major problem with this approach is that GIF images are expensive to generate and transport. Since the GIF conversion must occur on the fly, in response to each query (we later plan to investigate caching the most frequently used images), we have invested considerable effort in finding the most efficient means of performing this conversion. From the display, we can generate a pixmap. The pixmap is converted to an intermediate form, and an off-the-shelf C library completes the translation to GIF. This process is still slow for large images, however. We therefore looked for an alternative to keep the images as small as possible. Many of the EcoCyc displays contain large regions of nothing but text. If the text formatting in the CLIM display is not essential, our preference is to return the text directly rather than encapsulating it within a GIF image. This approach requires some minor changes to the EcoCyc program (though none that affect its operation in the original single-user mode). Because EcoCyc displays tend to break very cleanly into text and graphics regions however, this separation resulted in much faster response times for Web operation. Depending on the relative proportions of text and graphics in a display, we were able to decrease the time to generate Web displays by 10% to 85% (these percentages do not include the time required to transmit the data over the Internet to the client, which, because the GIF files are smaller, should also decrease). Table 1's before and after timings for typical object displays of several types show how performance improved after displays were divided into text and graphics regions.

To implement these performance enhancements, we defined several new classes of CLIM output records — a class for display portions that should be converted to GIF, another class for data that is exclusively text, and others (for titles, etc.). We altered the EcoCyc program to use these new classes of output records in place of

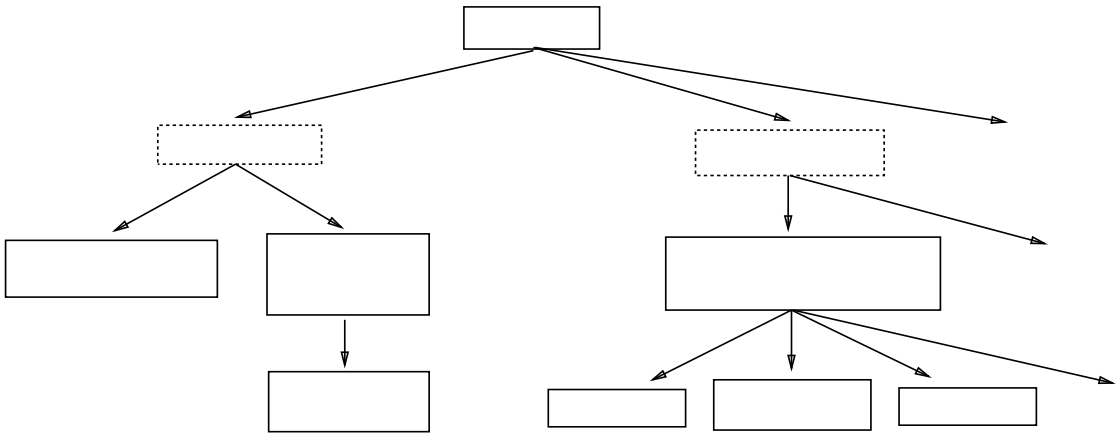

Figure 3: A fragment of an output record (OR) hierarchy

or in addition to the standard CLIM output record classes, whenever appropriate. Because these output record classes share all the same behaviors as the standard CLIM output record classes, the native CLIM display is unaffected; use of one of the new classes simply tells the Web server tool how to process the output record contents when generating Web output. Instead of converting the root output record to one large GIF file, the output record data structure is parsed recursively. Graphics output records are converted to inline mapped GIF images as described above. Text output records are converted to HTML instead. Presentations inside the text regions are converted directly to hypertext links. Figure 3 illustrates a small fragment of an output record hierarchy that might have been generated by the display in Figure 1. In addition to the information shown in the diagram, each output record also records the position of its contents in the CLIM display window, and any formatting constraints (color, text style, etc.). The Text-region and Graphics-region output records are added to support CWEST: again, they contain no additional information for the CLIM display but serve as signals for how their subtrees should be displayed on the Web. Figure 4 shows the display from Figure 1 as it appears in a WWW client window.

The final issue is that of state. The native EcoCyc application allows the user to set various display preferences that stay in effect as the user navigates from one display to another. HTTP is a stateless protocol, however, so all state must be encapsulated within each and every URL. When preferences are set (by means of an HTML fill-in form) a preference string is created that is appended to every hyperlink URL generated in the course of responding to a query. The preference string can be parsed to retrieve the set of preferences in a form that the EcoCyc program understands. In this way, state can be propagated from one display to another.

6 Discussion

One of the most surprising aspects of CWEST was just how easy it was to implement, requiring fewer than 150 person-hours of labor and 800 lines of Lisp code. That the EcoCyc user interface (14,000 lines of Lisp and CLIM code), which has been in development for the past two years, could be so easily adapted to support this new functionality is, we believe, a strong testament to the flexibility of Common Lisp and CLIM. When the EcoCyc project began we had not even heard of the WWW, so we were certainly not developing our application with this kind of extension explicitly in mind. Our experience demonstrates an important lesson for any software developer: if today's systems are to be successfully ported to tomorrow's technology, they must be developed using the most flexible tools available. Although CLIM has its disadvantages (for example, it can be slow, and would-be programmers experience a significant learning curve), we have difficulty imagining that a similar existing application developed using Motif, or even Tcl/Tk, could be so easily converted to run over the WWW.

We were fortunate that EcoCyc is well suited to the Web access style. Because most object displays are

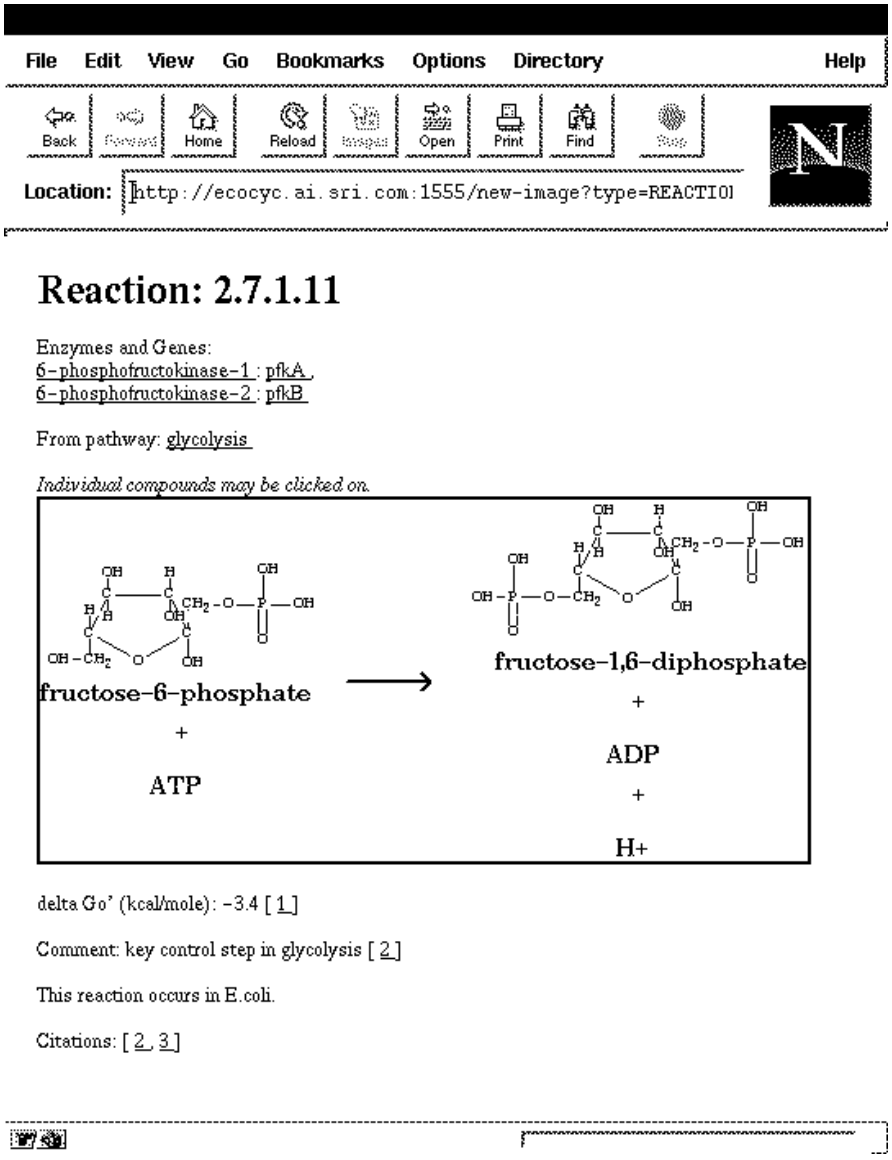

Figure 4: A sample EcoCyc display as it appears using the WWW client Netscape

independent of the path taken to reach them, we were able to keep to a minimum the amount of state that had to be encoded in each hyperlink. Most EcoCyc output goes to a single display pane (though EcoCyc can also operate in a mode that has multiple display panes), and very few multi-step operations, or interactive dialogs are involved. Those EcoCyc commands that do require a more complicated interaction style, particularly those that use menus or dialogs, are simply not currently supported by the Web implementation. A few important EcoCyc commands involving incrementally developed displays were rewritten slightly to be more compatible with our Web server implementation. Other CLIM developers may find varying degrees of success in porting their applications to the WWW, depending on how closely their applications' native interaction styles mesh with the Web interaction style. It should also be noted that the WWW supports only one type of gesture, a single mouse click, which might hamper developers who rely extensively on a three-button mouse.

Conversely, with EcoCyc running on the WWW, we have access to functionalities not available to EcoCyc running in native mode. For example, EcoCyc contains many references to data in other databases, such as the Medline and Genbank databases from NCBI. In native mode, when a user clicks on such a reference, a command is issued to fetch the data from the remote database and display it. In Web mode, however, we can create a

hyperlink to the Web page for that remote database entry. Now, in addition to viewing a single citation, the user can view all hyperlinks, such as related citations or keyword definitions, originating from that entry.

When formatting text regions in a display, a three way trade-off exists between the amount of formatting control exercised by the application, the complexity of the implementation, and speed of display. By converting the entire CLIM window to a GIF image, any text will be formatted exactly as it was in the original CLIM display, giving complete formatting control to the application programmer. Of course, this is also the slowest approach. Alternatively, by simply passing all text directly to the client as ASCII characters, the transfer will be fastest but all formatting control will be lost. The middle ground, which we have chosen, is to increase the complexity of the server implementation to recognize a reasonable set of formatting constraints and to ignore all others. Of the two ways to accomplish this task, we have chosen to define a few new classes of output records that will generate HTML formatting commands for titles, subtitles, hard line breaks, etc., while ignoring such factors as font, face (bold, italic, etc.) and color. The disadvantage of this approach is that it requires modification to the existing program to make use of the new output record classes. The more formatting control the programmer desires, the greater the number of output record classes that must be defined, and the more modification to the existing program is required. The other alternative would be to parse the CLIM output records more extensively to determine precisely which formatting options are in effect, where the line breaks fall, etc. Although this approach gives us access to all formatting options, its disadvantage is that parsing the CLIM internal output records requires delving into the internal CLIM data structures, which are not likely to be portable between vendors or versions. Also, not all CLIM formatting options can currently be converted to HTML (indentation, for example), so using this second approach to determine where line breaks should fall is not likely to be particularly effective. Perhaps the best alternative would be some combination of these two approaches.

7 Related Work

Mallery[3] has implemented a complete HTTP server in Common Lisp. This server has been used to extend an application that generates and processes email-based forms to also generate and process forms for the WWW. CLIM presentation methods have been written that specialize on the type of view, either email or HTML, so that the same generic method calls can handle both cases. This is another demonstration of how CLIM can be used to facilitate software reuse in converting an existing application to run over the WWW. The difference between Mallery's work and ours is primarily a difference in focus. His goal is to provide a general environment for rapid prototyping of Web applications and, more specifically, to develop seamless form processing over both email and the WWW, whereas our intent is to reuse CLIM applications over the Web by tightly coupling CLIM displays to Web input and output. Mallery also makes no attempt to handle CLIM graphics.

Thralls[4] has implemented a tool to convert existing Microsoft Windows applications to a series of HTML fill-in forms. He parses the resource files that describe an application's dialog boxes, menus, and other display items. The results are used to generate HTML output and C code that produces backend host programs, conforming to Common Gateway Interface (CGI) standards, to process the form data. This tool has been used to develop a Web version of a database access application. The data retrieved from the database can include both text and figures. However, the returned text and figures must be returned as stored in the database and cannot be generated dynamically, as they can using our application.

8 Conclusion

We have developed a Web server tool, CWEST, implemented in Common Lisp and CLIM, that facilitates the adaptation of an existing CLIM application to run over the WWW. Local CLIM displays can be broken into regions of text and graphics and translated to HTML and GIF, respectively. The CLIM output records for these displays can be parsed, and any presentations can be used to generate Web hyperlinks. We have successfully

used this tool to operate the EcoCyc program over the WWW, with very few changes to the underlying EcoCyc code. The work described here will facilitate the use of CLIM as a high-level programming tool for generating WWW applications and should prove useful to current CLIM developers, enabling them to maximize software reuse when porting their applications to the WWW.

9 Future Work

Our server tool currently supports a subset of CLIM functionality. It could conceivably be extended in many ways. For example, we might add facilities to form a display by collecting output from multiple CLIM panes. We could also integrate Mallery's work to translate CLIM menus and dialogs to HTML fill-in forms. Currently, a display is generated from scratch in response to each Web request. For many CLIM applications, however, many displays are generated in incremental fashion, with each new display varying in some small way from the one before. In this case, caching the output record or even the GIF file that is generated in response to one request, and reusing it for subsequent related requests might be more efficient. This may require implementation of a more elaborate form of session-tracking, with user login required. Even for EcoCyc, which does not require incrementally generated displays, we are interested in ways of speeding up GIF file creation and are considering caching GIF files for the most frequently accessed object displays.

It should be noted that CLIM provides a very rich range of possible user interactions, many of which cannot be efficiently supported with a Web-style of interaction as it is currently constituted. As Web technology grows and changes, we expect CWEST to expand in concert to support an ever-increasing range of CLIM functionalities.

Acknowledgments

We are grateful to Yvan Leclerc for his assistance on WWW issues. Colin Meldrum of Franz, Inc. was helpful in our efforts to speed up conversion of CLIM images to GIF. The final step in the conversion to GIF was accomplished using the *gd* graphics library from the Quest Protein Database Center. This work was partially supported by grant 1-R01-RR07861-01 from the National Center for Research Resources. The contents of this article are solely the responsibility of the authors and do not necessarily represent the official views of the National Institutes of Health.

References

[1] Franz, Inc., Berkeley, CA. *Common Lisp Interface Manager User Guide*, 1994.

[2] P. Karp. The EcoCyc user's guide. unpublished; see WWW URL `ftp://ftp.ai.sri.com/pub/papers/karp-ecocyc-guide.ps.Z`, 1994.

[3] John C. Mallery. A Common LISP hypermedia server. In *Proceedings of the First International Conference on the World-Wide Web*, 1994. `http://www.ai.mit.edu/projects/iiip/doc/cl-http/server-abstract.html`.

[4] Robert K. Thralls. Building HTML application systems: Converting existing MS-Windows applications to HTML. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994. `http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/WebProd/thralls/WhitePaper.html`.